REMARKS

This is a full and timely response to the outstanding final Office Action mailed June 16, 2004. Reconsideration and allowance of the application and pending claims are respectfully requested.

I. Claim Rejections - 35 U.S.C. § 102(e)

A. Rejection of Claims

Claims 23, 30, 36, 40, 46, 49, 54, and 63 have been rejected under 35 U.S.C. § 102(e) as allegedly being anticipated by *Grove* (U.S. Pat. No. 5,857,103). Applicant respectfully traverses this rejection.

B. Applicant's Claimed Invention

As provided in independent claims 23, 30, 36, 40, 46, 49, and 54, Applicant claims:

23. A method for passing information to a post-compile-time software application, comprising the steps of:

compiling a plurality of blocks of code, including finding one or more unused bits in an instruction in one of the plurality of blocks of code, and ***using the one or more unused bits to encode information without defining new instructions or opcodes***; and

communicating the information to the post-compile-time software application for use by the post-compile-time software application.

30. A system comprising:

a compiler that is configured to compile a plurality of blocks of code, the compiler including ***a code annotator*** that is ***configured to*** find one or more unused bits in an instruction in one of the plurality of blocks of code that are being compiled by the compiler, and to ***encode information in the one or more unused bits without defining new instructions or opcodes***, the information being configured ***to be used by a post-compile-time software application***; and

a processor configured to execute the compiler.

36. A system comprising:

means for compiling a plurality of blocks of code, including finding one or more unused bits in an instruction in one of the plurality of blocks of code, and ***using***

the one or more unused bits to pass information to a post-compile-time software application without defining new instructions or opcodes; and

means for executing the means for compiling and the post-compile time software application.

40. A method for compiling, comprising the steps of:

finding one or more unused bits in an instruction in one of a plurality of blocks of code that are being compiled; and

encoding information in the one or more unused bits without defining new instructions or opcodes, wherein the information is used by a post-compile-time software application.

46. A method for passing information to a post-compile-time software application, comprising the steps of:

compiling a plurality of blocks of code; and

during the step of compiling, finding one or more unused bits in an instruction in one of the plurality of blocks of code, ***wherein the one or more unused bits are used to pass information to the post-compile-time software application without defining new instructions or opcodes.***

49. A computer system comprising:

memory configured to store software;

a processor that is coupled to the memory and that is configured to execute software stored in the memory;

a compiler that is stored in the memory and that is configured to compile blocks of code, to find unused bits in an instruction in one of the blocks of code, ***and to encode information as a bit vector in the unused bits***; and

a post-compile-time software application that is stored in the memory and that is configured to use the information to modify the compiled blocks of code.

54. A method that is implemented by software, comprising the steps of:

storing a post-compile-time software application in memory that is coupled to a processor;

compiling blocks of code, including finding unused bits in an instruction in one of the blocks of code, ***and encoding information as a bit vector in the unused bits***; and

modifying the compiled blocks of code by the post-compile-time software application, wherein a configuration of the modified and compiled blocks of code is responsive to the information.

C. Discussion of the Rejection

It is axiomatic that "[a]nticipation requires the disclosure in a single prior art reference of *each element* of the claim under consideration." *W.L. Gore & Associates, Inc. v. Garlock, Inc.*, 721 F.2d 1540, 1554, 220 U.S.P.Q. 303, 313 (Fed. Cir. 1983) (emphasis added). Therefore, every claimed feature of the claimed invention must be represented in the applied reference to constitute a proper rejection under 35 U.S.C. § 102(e). In the present case, not every feature of the claimed invention is represented in the *Grove* reference.

Independent Claim 23

Grove does not disclose "using the one or more unused bits to encode information without defining new instructions or opcodes," or "communicating the information to the post-compile-time software application for use by the post-compile-time software application," as recited in independent claim 23. The final Office Action alleges with regard to the above features:

> Regarding claims 23, 40, 46 & 49 a method (Col. 14 to Col. 16 line 40), a system (Col. 16 line 38 to Col. 17 line 17), for passing information to a post-compile-time software application comprising the steps of:
>
> compiling a plurality of blocks of, and using the one or more unused bits to encode information (Grove, Col. 10 lines 10-15, also figure 6) without defining new instructions or opcodes (4:59-67, shows using unused bits to encode existing instructions and not new instructions);
>
> communicating the information to the post-compile-time software application (Grove, 9:58-60, refer to LUT/look up table, which is used by compiler, also see 13:10-35 for LUT and bit pattern).

The Office Action further notes in the Response to Arguments Section:

> Regarding Applicant's limitation of "without defining new instructions or opcodes...", as amended in independent claims 23, 30, 36, & 40, Examiner believes the prior art of record Groove still teaches this limitation. As set forth above and recited in Groove in 4:59 – 67, Groove shows using unused bits to encode existing instructions and not new instructions. Although, Groove does have a separate embodiment which does encoded new instructions Groove also additionally teaches using the unused bits to perform other operations see Groove, 3:45 – 50 and 4:60-65 also see 10:60-65.

Applicant respectfully disagrees with what the Office Action alleges *Grove* is teaching. The cited sections (Col. 14 to Col.17) include claim sets that describe systems and methods that create a target executable program for a processor, as listed below (with emphasis added):

> independent claim 1: A computer implemented method for creating a target executable program from the source code of a target computer program **for execution on a target processor**;
> independent claim 5: A method of **modifying an instruction set of a target processor**;
> independent claim 9: A method of compiling the source code of a target computer program into a target executable program **which optimizes the use of registers on a target processor**;
> independent claim 10: An apparatus for creating a target executable program from the source code of a target computer program **for execution on a target processor**;
> independent claim 13: a computer usable medium having computer readable program code tangibly embedded in said computer program product, the program code configured to cause the computer to create a target executable program from the source code of a target computer program **for execution on a target processor**;

None of these claims include the claimed feature "communicating the information to the post-compile-time software application for use by the post-compile-time software application." Applicant respectfully submits that the reference to the LUT and bit pattern in *Grove* sections 9:58-60 and 13:10-35 is misplaced. In sections 9:58-60 and 13:10-35, *Grove* discloses the following:

> The information contained in the LUT is also used by a compiler, described in detail below, to generate an executable module.
>
> At step 906, the method determines if an instruction from the first set of instructions has been associated with the virtual register. In one embodiment, the method compares the opcode of the instruction with bit patterns in the LUT. The LUT indicates whether the instruction is in the first or the second set of instructions. If at least one instruction from the first set of instructions is associated with the virtual register then the virtual register being processed is marked as poisoned (step 908). However, if only instructions from the second set of instructions are associated with the virtual register then the virtual register is marked as not being poisoned (step 910). At step 912, the method determines if additional virtual registers were allocated by the code generator portion of the compiler. If no more virtual registers were allocated then the poisoning method ends at step 916.

Applicant respectfully submits that these cited sections do not anticipate the claimed feature "communicating the information to the post-compile-time software application for use by the post-compile-time software application." Referring to section 9:58-60, if the LUT is indeed used by the compiler, it is being used during the pendency of compilation, not post-compilation.

Referring to section 13:10-35, Applicant is unclear as to why this section is being cited. Although the exact location of the LUT is not shown, it is understood to be accessed by the decoder 313 (or possibly even part of the decoder 313) for comparison of opcodes of each instruction with various bit patterns, a match indicating a valid instruction has been provided (see col. 8, lines 43-50). There is no indication that the LUT is passed or communicated to a software application, or more particularly, these cited sections do not disclose the feature of "communicating the information to the post-compile-time software application for use by the post-compile-time software application," as recited in claim 23.

With regard to the claimed feature of "using the one or more unused bits to encode information without defining new instructions or opcodes," as recited in independent claim 23, Applicant respectfully submits that the cited sections do not disclose this feature. Section 4:59-67 provides as follows:

> In one embodiment, the prevent invention uses one or more unused bits in the first set of instructions to distinguish instructions in the first set of instructions from instructions in the second set of instructions.
>
> In yet another embodiment, the invention provides a method and apparatus for compiling the source code of a target computer program into a target executable program which optimizes the use of registers on the target processor described above.

From this cited portion, the processor apparently would be able to use the distinction to know whether to address the first set of instructions or the second set of instructions. Such purpose would not be used to communicate "the information to the post-compile-time software application for use by the post-compile-time software application," as recited in claim 23.

Further, the second set of instructions does not appear to be accomplished "without defining new instructions or opcodes," also as recited in claim 23. *Grove* discloses an apparatus and method that can address a second set of registers without adding width to the instructions (see col. 3, lines 45-50). It performs this function by either "modifying" or "creating" new instructions to handle the second set. The use of "modifying" is simply a matter of semantics, and in fact, since the second set of instructions is a subset of the first set of instructions (col. 9, lines 47-53), it is not really accurate to state that the first set is modified, as it retains its form with the addition of a second set of instructions. In fact, whether a new instruction is created as described in col. 10, or an existing instruction is modified as described in col. 9, new instructions are indeed defined, otherwise it does not make sense to modify the LUT (which contains all valid executable instructions), as described in col. 9, lines 53-60. However, it is important to note that the LUT needs to be modified (step 414 in the prior cited section). The LUT includes a list of all bit patters stored in the opcode portion of each valid instruction (col. 8, lines 45 and 46). Apparently, the second instruction set is not part of that list initially. Modification of the first instruction set requires the LUT to be updated to recognize the modified second instruction set. In that sense, the second instruction set is "new" (i.e., created from the first instruction set). Thus, *Grove* does not meet the claimed feature, "using the one or more unused bits to encode information without defining new instructions or opcodes," as recited in claim 23.

For at least the reasons stated above, *Grove* does not anticipate claim 23, and therefore the Applicant respectfully requests that the rejection to claim 23 be withdrawn.

Because independent claim 23 is allowable over *Grove*, dependent claims 24-29 and 47-48 are allowable as a matter of law for at least the reason that the dependent claims 24-29 and 47-48 contain all the elements of their respective base claim. See, e.g., *In re Fine*, 837 F.2d 1071 (Fed. Cir. 1988).

Independent Claim 30

Grove does not disclose "a code annotator that is configured to find one or more unused bits in an instruction in one of the plurality of blocks of code that are being compiled by the compiler, and to encode information in the one or more unused bits without defining new instructions or opcodes, the information being configured to be used by a post-compile-time software application," as recited in independent claim 30.

Applicant respectfully disagrees with what the Office Action alleges *Grove* is teaching. The cited sections (Col. 14 to Col.17) include claim sets that describe systems and methods that create a target executable program for a processor, and not for use by "a post-compile-time software application," as recited in claim 30.

Also, if the LUT is indeed used by the compiler, it is being used during the pendency of compilation, not post-compilation. There is no indication that the LUT is passed or communicated to a software application, or more particularly, these cited sections do not disclose the feature of "the information being configured to be used by a post-compile-time software application," as recited in claim 30.

With regard to the claimed feature of "a code annotator that is configured to find one or more unused bits in an instruction in one of the plurality of blocks of code that are being compiled by the compiler, and to encode information in the one or more unused bits without defining new instructions or opcodes," as recited in independent claim 30, Applicant respectfully submits that the cited sections do not disclose these features. From the cited portions, the processor apparently would be able to use the distinction between the first and second instruction to know whether to address the first set of instructions or the second set of instructions. Such purpose would not be used to communicate "the information being configured to be used by a post-compile-time software application," as recited in independent claim 30.

Further, the second set of instructions does not appear to be accomplished "without defining new instructions or opcodes," also as recited in claim 30. *Grove* discloses an apparatus and method that can address a second set of registers without adding width to the instructions. It performs this function by either "modifying" or "creating" new instructions to handle the second set. The use of "modifying" is simply a matter of semantics, and in fact, since the second set of instructions is a subset of the first set of instructions, it is not really accurate to state that the first set is modified, as it appears to retain its form with the addition of a second set of instructions. In fact, whether a new instruction is created as described in col. 10, or an existing instruction is modified as described in col. 9, new instructions are indeed defined, otherwise it does not make sense to modify the LUT. The LUT includes a list of all bit patterns stored in the opcode portion of each valid instruction. Apparently, the second instruction set is not part of that list initially. Modification of the first instruction set requires the LUT to be updated to recognize the modified second instruction set. In that sense, the second instruction set is "new" (*i.e.*, created from the first instruction set). Thus, *Grove* does not meet the claimed feature, "a code annotator that is configured to find one or more unused bits in an instruction in one of the plurality of blocks of code that are being compiled by the compiler, and to encode information in the one or more unused bits without defining new instructions or opcodes," as recited in claim 30.

For at least the reasons stated above, *Grove* does not anticipate claim 30, and therefore the Applicant respectfully requests that the rejection to claim 30 be withdrawn.

Because independent claim 30 is allowable over *Grove*, dependent claims 31-35 are allowable as a matter of law.

Independent Claim 36

Grove does not disclose "using the one or more unused bits to pass information to a post-compile-time software application without defining new instructions or opcodes" or "means for executing the means for compiling and the post-compile time software application" as recited in independent claim 36. Applicant respectfully disagrees with what the Office Action alleges *Grove* is teaching. The cited sections (Col. 14 to Col.17) include claim sets that describe systems and methods that create a target executable program for a processor, and not for execution by "means for executing the means for compiling and the post-compile time software application," as recited in claim 36.

Also, if the LUT is indeed used by the compiler, it is being used during the pendency of compilation, not post-compilation. There is no indication that the LUT is passed or communicated to a "means for executing the means for compiling and the post-compile time software application," as recited in claim 36.

With regard to the claimed feature of "using the one or more unused bits to pass information to a post-compile-time software application without defining new instructions or opcodes," as recited in independent claim 36, Applicant respectfully submits that the cited sections do not disclose these features. From the cited portions, the processor apparently would be able to use the distinction between the first and second instruction to know whether to address the first set of instructions or the second set of instructions. Such purpose would not be "using the one or more unused bits to pass information to a post-compile-time software application without defining new instructions or opcodes," as recited in independent claim 36."

Further, the second set of instructions does not appear to be accomplished "without defining new instructions or opcodes," also as recited in claim 36. *Grove* discloses an apparatus and method that can address a second set of registers without adding width to the instructions. It performs this function by either "modifying" or "creating" new instructions to handle the second set. The use of "modifying" is simply a matter of semantics, and in fact, since the second set of instructions is a subset of the first set of instructions, it is not really accurate to state that the first set is modified, as it appears to retain its form with the addition of a second set of instructions. In fact, whether a new instruction is created as described in col. 10, or an existing instruction is modified as described in col. 9, new instructions are indeed defined, otherwise it does not make sense to modify the LUT. The LUT includes a list of all bit patterns stored in the opcode portion of each valid instruction. Apparently, the second instruction set is not part of that list initially. Modification of the first instruction set requires the LUT to be updated to recognize the modified second instruction set. In that sense, the second instruction set is "new" (i.e., created from the first instruction set). Thus, *Grove* does not meet the claimed feature, "using the one or more unused bits to pass information to a post-compile-time software application without defining new instructions or opcodes" as recited in independent claim 36."

For at least the reasons stated above, *Grove* does not anticipate claim 36, and therefore the Applicant respectfully requests that the rejection to claim 36 be withdrawn.

Because independent claim 36 is allowable over *Grove*, dependent claims 37-39 are allowable as a matter of law.

Independent Claim 40

Grove does not disclose "encoding information in the one or more unused bits without defining new instructions or opcodes, wherein the information is used by a post-compile-time software application," as recited in independent claim 40. Applicant respectfully disagrees with what the Office Action alleges *Grove* is teaching. The cited sections (Col. 14 to Col.17) include claim sets that describe systems and methods that create a target executable program for a processor, and not for use by "a post-compile-time software application," as recited in claim 40.

Also, if the LUT is indeed used by the compiler, it is being used during the pendency of compilation, not post-compilation. There is no indication that the LUT is passed or communicated to a software application, or more particularly, these cited sections do not disclose the feature of "wherein the information is used by a post-compile-time software application," as recited in claim 40.

With regard to the claimed feature of "encoding information in the one or more unused bits without defining new instructions or opcodes," as recited in independent claim 40, Applicant respectfully submits that the cited sections do not disclose this feature. From the cited portions, the processor apparently would be able to use the distinction between the first and second instruction to know whether to address the first set of instructions or the second set of instructions. Such purpose would not be used to communicate "wherein the information is used by a post-compile-time software application," as recited in independent claim 40.

Further, the second set of instructions does not appear to be accomplished "without defining new instructions or opcodes," also as recited in claim 40. *Grove* discloses an apparatus and method that can address a second set of registers without adding width to the instructions. It performs this function by either "modifying" or "creating" new instructions to handle the second set. The use of "modifying" is simply a matter of semantics, and in fact, since the second set of instructions is a subset of the first set of instructions, it is not really accurate to state that the first set is modified, as it appears to retain its form with the addition of a second set of instructions. In fact, whether a new instruction is created as described in col. 10, or an existing instruction is modified as described in col. 9, new instructions are indeed defined, otherwise it does not make sense to modify the LUT. The LUT includes a list of all bit patterns stored in the opcode portion of each valid instruction. Apparently, the second instruction set is not part of that list initially. Modification of the first instruction set requires the LUT to be updated to recognize the modified second instruction set. In that sense, the second instruction set is "new" (*i.e.*, created from the first instruction set). Thus, *Grove* does not meet the claimed feature, "encoding information in the one or more unused bits without defining new instructions or opcodes," as recited in claim 40.

For at least the reasons stated above, *Grove* does not anticipate claim 40, and therefore the Applicant respectfully requests that the rejection to claim 40 be withdrawn.

Because independent claim 40 is allowable over *Grove*, dependent claims 41-45 are allowable as a matter of law.

Independent Claim 46

Grove does not disclose "wherein the one or more unused bits are used to pass information to the post-compile-time software application without defining new instructions or opcodes," as recited in independent claim 46. Applicant respectfully disagrees with what the Office Action alleges *Grove* is teaching. The cited sections (Col. 14 to Col.17) include claim sets that describe systems and methods that create a target executable program for a processor, and not for use by "a post-compile-time software application," as recited in claim 46.

Also, if the LUT is indeed used by the compiler, it is being used during the pendency of compilation, not post-compilation. There is no indication that the LUT is passed or communicated to a software application, or more particularly, these cited sections do not disclose the feature of "wherein the one or more unused bits are used to pass information to the post-compile-time software application," as recited in claim 46.

With regard to the claimed feature of "wherein the one or more unused bits are used to pass information to the post-compile-time software application without defining new instructions or opcodes," as recited in independent claim 46, Applicant respectfully submits that the cited sections do not disclose these features. From the cited portions, the processor apparently would be able to use the distinction between the first and second instruction to know whether to address the first set of instructions or the second set of instructions. Such purpose would not be used to communicate "wherein the one or more unused bits are used to pass information to the post-compile-time software application," as recited in independent claim 46.

Further, the second set of instructions does not appear to be accomplished "without defining new instructions or opcodes," also as recited in claim 46. *Grove* discloses an apparatus and method that can address a second set of registers without adding width to the instructions. It performs this function by either "modifying" or "creating" new instructions to handle the second set. The use of "modifying" is simply a matter of semantics, and in fact, since the second set of instructions is a subset of the first set of instructions, it is not really accurate to state that the first set is modified, as it appears to retain its form with the addition of a second set of instructions. In fact, whether a new instruction is created as described in col. 10, or an existing instruction is modified as described in col. 9, new instructions are indeed defined, otherwise it does not make sense to modify the LUT. The LUT includes a list of all bit patterns stored in the opcode portion of each valid instruction. Apparently, the second instruction set is not part of that list initially. Modification of the first instruction set requires the LUT to be updated to recognize the modified second instruction set. In that sense, the second instruction set is "new" (*i.e.*, created from the first instruction set). Thus, *Grove* does not meet the claimed feature, "wherein the one or more unused bits are used to pass information to the post-compile-time software application without defining new instructions or opcodes," as recited in claim 46.

For at least the reasons stated above, *Grove* does not anticipate claim 46, and therefore the Applicant respectfully requests that the rejection to claim 46 be withdrawn.

Independent Claim 49

Grove does not disclose "a post-compile-time software application that is stored in the memory and that is configured to use the information to modify the compiled blocks of code," as recited in independent claim 49. Applicant respectfully disagrees with what the Office Action alleges *Grove* is teaching. The cited sections (Col. 14 to Col.17) include claim sets that describe systems and methods that create a target executable program for a processor, and not "a post-compile-time software application that is stored in the memory and that is configured to use the information to modify the compiled blocks of code," as recited in claim 49.

Also, if the LUT is indeed used by the compiler, it is being used during the pendency of compilation, not post-compilation. There is no indication that the LUT is passed or communicated to a software application, or more particularly, these cited sections do not disclose "a post-compile-time software application that is stored in the memory and that is configured to use the information to modify the compiled blocks of code," as recited in claim 49.

Also, *Grove* does not disclose "and to encode information as a bit vector in the unused bits," as recited in claim 49. Although not addressed with reference to claim 49, the Office Action does address a similar limitation found in claim 54 through recitation of col. 13, lines 20-55, reproduced below:

> At step 906, the method determines if an instruction from the first set of instructions has been associated with the virtual register. In one embodiment, the method compares the opcode of the instruction with bit patterns in the LUT. The LUT indicates whether the instruction is in the first or the second set of instructions. If at least one instruction from the first set of instructions is associated with the virtual register then the virtual register being processed is marked as poisoned (step 908). However, if only instructions from the second set of instructions are associated with the virtual register then the virtual register is marked as not being poisoned (step 910). At step 912, the method determines if additional virtual registers were allocated by the code generator portion of the compiler. If no more virtual registers were allocated then the poisoning method ends at step 916. Conversely, if additional virtual registers were allocated then the next virtual register is retrieved at step 914 and processed in a similar fashion starting at step 906. This process is repeated for all virtual registers until all virtual registers allocated by the compiler have been processed and marked accordingly.
>
> Referring back to FIG. 8, at step 812 the backend portion of the compiler determines which instructions to generate and how to allocate physical registers on the processor to the virtual registers assigned to the intermediate instructions. Initially, the backend receives an intermediate instruction and determines if the intermediate instruction should be substituted with an instruction in the first set of instructions or the second set of instructions. If the

> intermediate instruction is associated with a poisoned virtual register then the backend will substitute an instruction from the first set of instructions listed in the LUT and allocate one or more registers from the first set of registers. However, if the intermediate instruction is not associated with a poisoned register then the back end will substitute an instruction from the second set of instructions listed in the LUT and allocate one or more registers from the second set of registers.

Applicant fails to see the where the claimed feature "and to encode information as a bit vector in the unused bits" is used in this cited portion, and respectfully requests clarification on this aspect of the Office Action.

For at least the reasons stated above, *Grove* does not anticipate claim 49, and therefore the Applicant respectfully requests that the rejection to claim 49 be withdrawn.

Because independent claim 49 is allowable over *Grove*, dependent claims 50-53 are allowable as a matter of law.

Independent Claim 54

Grove does not disclose "storing a post-compile-time software application in memory that is coupled to a processor," or modifying the compiled blocks of code by the post-compile-time software application," as recited in independent claim 54. Applicant respectfully disagrees with what the Office Action alleges *Grove* is teaching. The cited sections (Col. 14 to Col.17) include claim sets that describe systems and methods that create a target executable program for a processor, and not "storing a post-compile-time software application in memory that is coupled to a processor," or modifying the compiled blocks of code by the post-compile-time software application," as recited in claim 54.

Also, if the LUT is indeed used by the compiler, it is being used during the pendency of compilation, not post-compilation. There is no indication that the LUT is passed or communicated to a software application, or more particularly, these cited sections do not disclose "storing a post-compile-time software application in memory that is coupled to a processor," or modifying the compiled blocks of code by the post-compile-time software application," as recited in claim 54.

Applicant fails to see the where the claimed feature "and encoding information as a bit vector in the unused bits" is used in this cited portion, and respectfully requests clarification on this aspect of the Office Action.

For at least the reasons stated above, *Grove* does not anticipate claim 54, and therefore the Applicant respectfully requests that the rejection to claim 54 be withdrawn.

Because independent claim 54 is allowable over *Grove*, dependent claims 55-58 are allowable as a matter of law.

II. Claim Rejections - 35 U.S.C. § 103(a)

A. Rejection of Claims

Claims 24-29, 31-35, 37-39, 41-45, 47, 48, 50 & 59-62 have been rejected under 35 U.S.C. § 103(a) as allegedly being unpatentable over *Grove* as applied in claim 23 and in view of *Hayashi* (U.S. Pat. No. 5,828,866). Applicant respectfully traverses this rejection.

B. Applicant's Claimed Invention

As provided in independent claims 59 and 61, Applicant claims:

59. A computer system comprising:
a compiler configured to compile blocks of code, to **find unused bits in a NOP instruction in one of the blocks of code, and to encode information as a bit vector in the unused bits, wherein the information identifies whether certain registers in said one of the blocks of code are live**; and
a dynamic optimizer that is configured to optimize the compiled blocks of code, wherein a configuration of the optimized and compiled blocks of code is responsive to the information.

61. A method comprising:
compiling blocks of code, including **finding unused bits in a NOP instruction in one of the blocks of code, and encoding information as a bit vector in the unused bits, wherein the information identifies whether certain registers in said one of the blocks of code are live**; and
optimizing the compiled blocks of code, wherein a configuration of the optimized and compiled blocks of code is responsive to the information.

C. Discussion of the Rejection

As acknowledged by the Court of Appeals for the Federal Circuit, the U.S. Patent and Trademark Office ("USPTO") has the burden under section 103 to establish a proper case of obviousness by showing some objective teaching in the prior art or generally available knowledge of one of ordinary skill in the art that would lead that individual to the claimed invention. See *In Re Fine*, 837 F.2d 1071, 5 U.S.P.Q.2d 1596, 1598 (Fed. Cir. 1988). Accordingly, to make a proper case for obviousness, there must be some prior art teaching or established knowledge that would suggest to a person having ordinary skill in the pertinent art to fill the voids apparent in the applied reference. It is respectfully asserted that no such case has been made in the outstanding Office Action.

Dependent Claims 24-29, 31-35, 37-39, 41-45, 47, 48, 50

In that *Grove* fails to disclose claim limitations found in independent claims 23, 30, 40, 46, 49, and 54, and that *Hayashi* does not remedy these deficiencies, Applicant respectfully submits that neither *Grove* nor *Hayashi*, alone or in combination, disclose, teach, or suggest all the limitations in dependent claims 24-29, 31-35, 37-39, 41-45, 47, 48, and 50, which contain all of the features of their respective allowable independent claims.

Independent Claim 59

Neither *Grove* nor *Hayashi*, alone or in combination, disclose, teach, or suggest the features "find unused bits in a NOP instruction in one of the blocks of code, and to encode information as a bit vector in the unused bits, wherein the information identifies whether certain registers in said one of the blocks of code are live," or "a dynamic optimizer that is configured to optimize the compiled blocks of code," as recited in claim 59. While *Hayashi* does disclose a register allotting process and instruction scheduling process, Applicant respectfully submits that *Hayashi* does not use any portion of the NOP instruction to store or carry register information. As evidenced by every example and the entirety of the *Hayashi* disclosure, the system of *Hyashi* never uses the NOP instruction to hold a code usage register. For example, the table (col. 15, line 57 through col. 16, line 24) shows twelve NOP instructions. However, Applicant respectfully submits that none of the twelve NOP instructions are associated with any register information.

Similarly, the table (col. 18, line 46 through col. 19, line 8) shows four NOP instructions, which are all devoid of register information. In similar fashion, Applicant respectfully submits that every NOP instruction in every one of *Hayashi*'s examples is devoid of register information. Thus, neither *Grove* nor *Hayashi*, alone or in combination, disclose, teach, or suggest the emphasized features "find unused bits in a NOP instruction in one of the blocks of code, and to encode information as a bit vector in the unused bits, wherein the information identifies whether certain registers in said one of the blocks of code are live."

Further, neither *Grove* nor *Hayashi*, alone or in combination, disclose, teach, or suggest the emphasized feature "a dynamic optimizer that is configured to optimize the compiled blocks of code." In other words, unlike the optimizer disclosed in *Grove*, the optimization disclosed in claim 59 occurs post-compilation. Thus, Applicant respectfully requests that the rejection to claim 59 be withdrawn.

Because independent claim 59 is allowable over *Grove* and *Hayashi*, dependent claim 60 is allowable as a matter of law.

Independent Claim 61

Neither *Grove* nor *Hayashi*, alone or in combination, disclose, teach, or suggest the features "finding unused bits in a NOP instruction in one of the blocks of code, and encoding information as a bit vector in the unused bits, wherein the information identifies whether certain registers in said one of the blocks of code are live," or "optimizing the compiled blocks of code," as recited in independent claim 61. Applicant respectfully submits that *Hayashi* does not use any portion of the NOP instruction to store or carry register information. Thus, neither *Grove* nor *Hayashi*, alone or in combination, disclose, teach, or suggest the emphasized features "finding unused bits in a NOP instruction in one of the blocks of code, and encoding information as a bit vector in the unused bits, wherein the information identifies whether certain registers in said one of the blocks of code are live."

Further, neither *Grove* nor *Hayashi*, alone or in combination, disclose, teach, or suggest the emphasized feature "optimizing the compiled blocks of code." In other words, unlike the optimizer disclosed in Grove, the optimization disclosed in claim 61 occurs post-compilation. Thus, Applicant respectfully requests that the rejection to claim 61 be withdrawn.

Because independent claim 61 is allowable over *Grove* and *Hyashi*, dependent claims 62-63 are allowable as a matter of law.

CONCLUSION

Applicant respectfully submits that Applicant's pending claims are in condition for allowance. Favorable reconsideration and allowance of the present application and all pending claims are hereby courteously requested. If, in the opinion of the Examiner, a telephonic conference would expedite the examination of this matter, the Examiner is invited to call the undersigned attorney at (770) 933-9500.

Respectfully submitted,

[illegible]

David Rodack
Registration No. 47,034

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postage prepaid, in an envelope addressed to: Assistant Commissioner for Patents, Alexandria, Virginia 22313-1450, on 8-16-04.

Mary Meegan
Signature